

# **Results from the *mimic-cafk* test**

**Adam Lyon, September 7, 2005**

## **1 Introduction**

A SAM test was performed to determine the rate of file deliveries possible on a test CAF system.

Andrew made a change to the station in the middle of the test that made significant improvement!

## Appendix A R Session

R is an open source statistical analysis software package that allows for very easy analysis of data in databases and text files. I wrote a "notebook" style package that allows one to use R from within Microsoft Word. Below is the notebook providing all of the code and results for this document.

### A.1 Connect to R and initialize

Connect to R running locally on my laptop.

```
<R0> #connect port 6101 timeout 20
R is using work directory /Users/adam/work/projects/cdfSamTests/mimic-
cafk
Set up graphics
<R1> library(lattice)

<R2> trellis.par.set(col.whitebg())

<R3> fontsize = trellis.par.get("fontsize"); fontsize$text=16 ;
      fontsize$points=6 ; trellis.par.set("fontsize", fontsize)
```

I have a helper function that makes putting graphics into Word easy.

```
<R4> mp
function (plotExpr, file, height = 7, width = 7, res = 72 * 3)
{
  bitmap(file, "pngalpha", height = height, width = width,
         res = res, pointsize = 10)
  r = eval(plotExpr)
  if (class(r) == "trellis")
    print(r)
  invisible(dev.off())
}
<environment: namespace:RemoteRSOAP>
```

### A.2 Running job output

Doug's python script loops over getting the next file, waiting thirty seconds to simulate processing, and then releasing the file. A log file records the time of the get next file, the duration of the command, the pnfs name of the file, and the time and duration of the release file command.

Let's read this information into R.

```

<R5> d = read.table("out.log", header=T)

<R6> d[1:5,]
   job segment getDate getTime getDur fileNum
1 15407       1 20050906 102818 274.822      1
2 15407       1 20050906 103309 10.657      2
3 15407       1 20050906 103336 1.582      3
4 15407       1 20050906 103353 13.606      4
5 15407       1 20050906 103424 16.463      5

pnfs
1
dcap://cdfdca2.fnal.gov:25149/pnfs/fnal.gov/usr/cdfen/filesets/GJ/GJ22/
GJ2275/GJ2275.0/xd025b54.00c8bhd0
2
dcap://cdfdca2.fnal.gov:25149/pnfs/fnal.gov/usr/cdfen/filesets/GJ/GJ22/
GJ2283/GJ2283.0/xd025c1e.004ebhd0
3
dcap://cdfdca2.fnal.gov:25149/pnfs/fnal.gov/usr/cdfen/filesets/GJ/GJ22/
GJ2284/GJ2284.0/xd025b54.0283bhd0
4
dcap://cdfdca2.fnal.gov:25149/pnfs/fnal.gov/usr/cdfen/filesets/GJ/GJ22/
GJ2284/GJ2284.0/xd025a62.00b8bhd0
5
dcap://cdfdca2.fnal.gov:25149/pnfs/fnal.gov/usr/cdfen/filesets/GJ/GJ22/
GJ2281/GJ2281.0/xd025a44.0347bhd0

   relDate relTime relDur
1 20050906 103308 1.094
2 20050906 103335 1.059
3 20050906 103353 0.836
4 20050906 103422 2.337
5 20050906 103456 0.918

```

Convert dates and times into POSIX times that R can deal with...

```
<R17> d$getTime = paste(d$getDate, d$getTime)
```

```
<R18> d$relDateTime = paste(d$relDate, d$relTime)
```

```
<R19> d$getPTime = as.POSIXct( strptime( d$getTime, "%Y%m%d %H%M%S"
) )
```

```
<R21> d$relPTime = as.POSIXct( strptime( d$relDateTime, "%Y%m%d %H%M%S"
) )
```

We can make the delivery time from the get time plus the duration

```
<R22> d$delPTime = d$getPTime + d$getDur
```

What is the range of the test?

```
<R23> testEdges = c(min(d$getPTime), max(d$relPTime, na.rm=T)) ;  
      testEdges
```

```
[1] "2005-09-06 10:27:14 CDT" "2005-09-06 12:09:39 CDT"
```

```
<R24> testTime = diff(testEdges) ; testTime  
Time difference of 1.706944 hours
```

```
<R25> prettyEdges = c(testEdges[1]-60*60, testEdges[2]+60*60)
```

### A.2.1 Basics

How many files deliveries were attempted?

```
<R26> nrow(d)  
[1] 16000
```

How many failed on the get end?

```
<R27> sum(is.na(d$getDur))  
[1] 0
```

How many failed on the release end?

```
<R28> sum(is.na(d$relDur))  
[1] 0
```

Merge job and segment numbers

```
<R29> d$jobseg = paste(d$job, d$segment)
```

What were the jobs and segments?

```
<R30> d$jobseg[ is.na(d$relDur) ]  
character(0)  
  
<R31> #var successfulDeliveries = nrow(d)  
16000
```

How come this isn't 40,000?

Look up the maximum file number for each job and segment.

```
<R32> largestFNum = tapply(d$fileNum, d$jobseg, max)
```

How many did not get all 40 files?

```
<R33> length( largestFNum[ largestFNum < 40 ] )
```

```
[1] 0
```

Hmmm.

```
<R34> notDone = largestFNum[ largestFNum < 40 ]
```

```
<R35> notDone[1:3]  
<NA> <NA> <NA>  
 NA   NA   NA
```

Will have to look these up!

### A.2.2 File delivery rate

```
<R36> fileRatePerDay = nrow(d)/(as.numeric(testTime))*24;  
      fileRatePerDay  
[1] 224963.4
```

In a perfect world, what should be the mean wait time?

```
<R37> filesPerSeg = nrow(d) / length(unique(d$jobseg)) ; filesPerSeg  
[1] 40
```

Assume that every segment runs the entire length of the test. Therefore (in minutes),

```
<R38> meanWaitTime = as.numeric(testTime)*60 / filesPerSeg ;  
      meanWaitTime  
[1] 2.560417
```

How many seconds per file?

```
<R39> secondsPerFile = as.numeric(testTime)*60*60 / nrow(d) ;  
      secondsPerFile  
[1] 0.3840625
```

### A.2.3 Number of segments running

Let's plot how many segments were running at a given time.

A segment turns on when it does its first "get next file". It turns off when it does its last release.

How many segments are there?

```
<R40> length( unique( d$jobseg ) )  
[1] 400
```

So we have a record of all 1000 segments. Good!

Segment starts when the first file is requested

```
<R41> segStart = data.frame( time=d$getPTime[d$fileNum==1], adj=1 )

<R42> segEnd = data.frame( time=d$relPTime[d$fileNum==40], adj=-1 )

<R43> segs = rbind(segStart, segEnd)

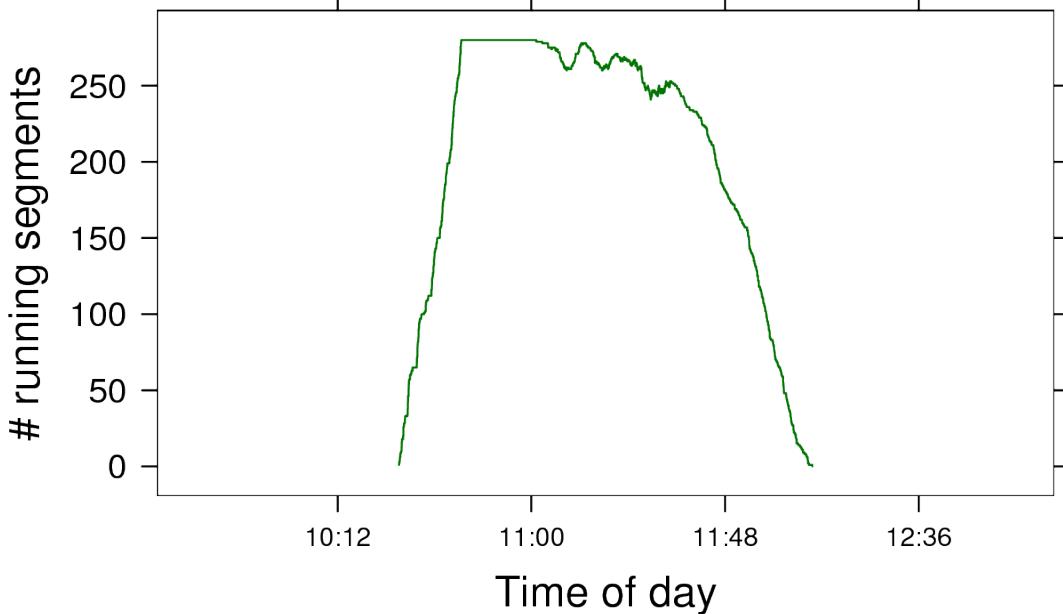
<R44> segs = segs[ order(segs$time), ]

<R45> segs$count = cumsum(segs$adj)

<R46> segs[1:10,]
      time adj count
48 2005-09-06 10:27:14   1     1
152 2005-09-06 10:27:14   1     2
80 2005-09-06 10:27:16   1     3
8 2005-09-06 10:27:20   1     4
64 2005-09-06 10:27:22   1     5
369 2005-09-06 10:27:23   1     6
176 2005-09-06 10:27:28   1     7
120 2005-09-06 10:27:30   1     8
32 2005-09-06 10:27:31   1     9
392 2005-09-06 10:27:42   1    10

<R47> mp(
  xyplot( segs$count ~ segs$time, type="s",
          main="Number of running segments",
          xlab="Time of day",
          ylab="# running segments",
          scales=list(x=list(tick.number=6, cex=0.6)),
          xlim=prettyEdges ),
  "nsegs.png", h=4, w=6 )
#with graphics nsegs.png timeout 120
```

## Number of running segments



### A.2.4 Number of gets and deliveries

Let's count up how many get file requests and deliveries there were per hour

```
<R48> getsPerHourCuts = cut( d$getPTime, "hours" )

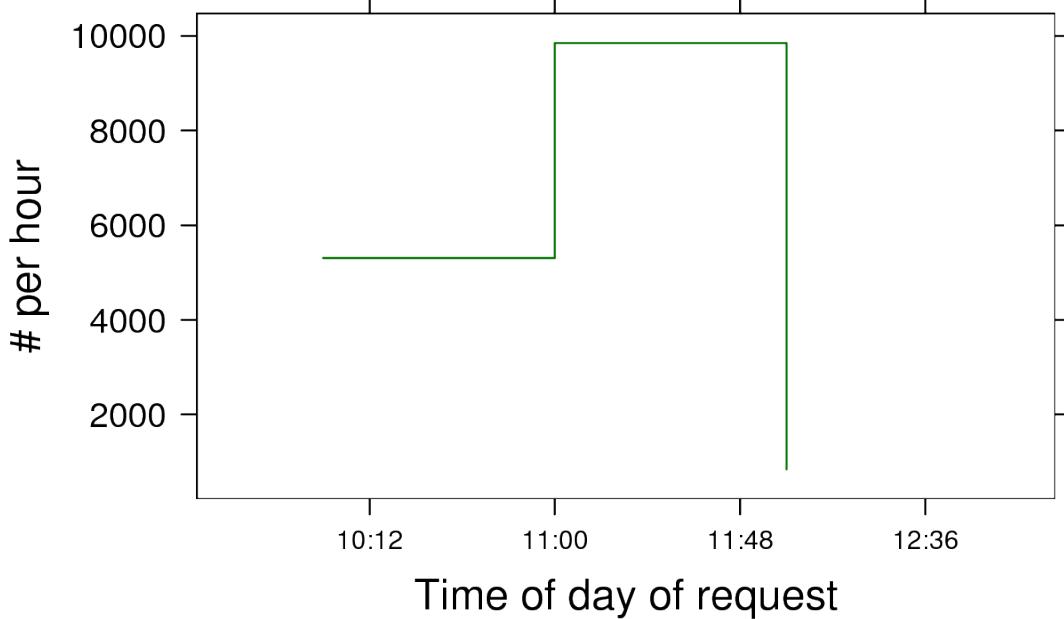
<R49> delsPerHourCuts = cut( d$delPTime, "hours" )

<R50> getsPerHour = tabulate(getsPerHourCuts)

<R51> delsPerHour = tabulate(delsPerHourCuts)

<R52> mp(
  xyplot( getsPerHour ~ as.POSIXct(levels(getsPerHourCuts)),
    main="Get file requests",
    xlab="Time of day of request",
    ylab="# per hour", type="s", xlim=prettyEdges,
    scales=list(x=list(tick.number=6, cex=0.6)))
),
"getsPerHour.png", h=4, w=6
)
#with graphics getsPerHour.png timeout 60
```

## Get file requests



```
<R53> mp(
  xyplot( delsPerHour ~ as.POSIXct(levels(delsPerHourCuts)),
    main="File deliveries",
    xlab="Time of day of delivery", xlim=prettyEdges,
    ylab="# per hour", type="s",
    scales=list(x=list(tick.number=6, cex=0.6))
  ),
  "delsPerHour.png", h=4, w=6
)
#with graphics delsPerHour.png timeout 60
```

## File deliveries



Let's just get a cumulative plot of when deliveries occurred.

```
<R54> deliveries = data.frame( time=d$delPTime, adj=1 )

<R55> deliveries = deliveries[ order(deliveries$time), ]

<R56> deliveries$count = cumsum(deliveries$adj)

<R57> nrow(deliveries)
[1] 16000

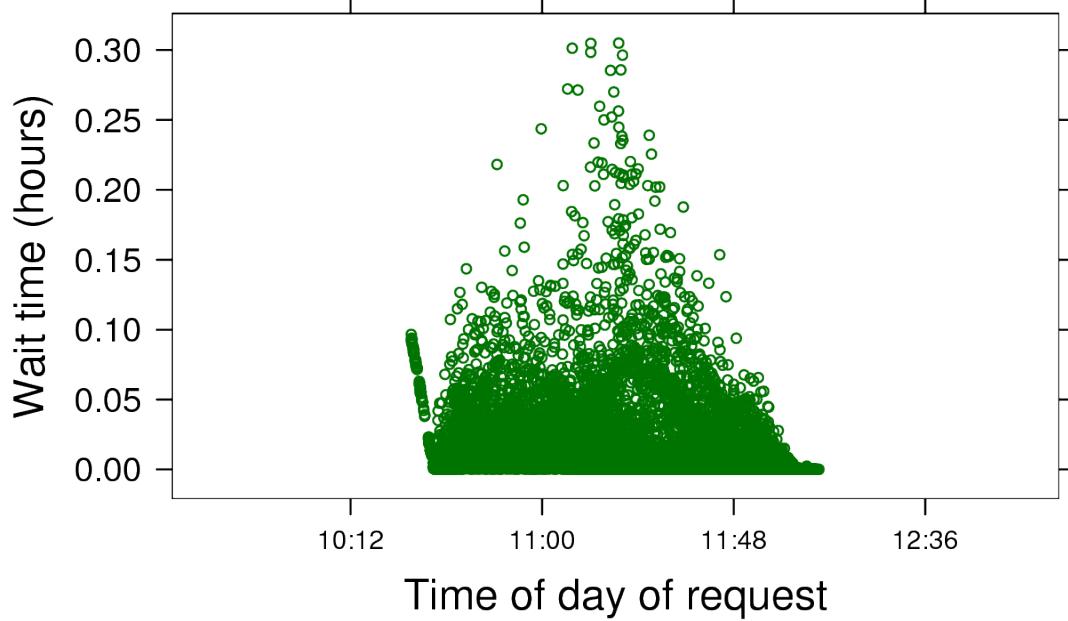
<R59> mp(
  xyplot( deliveries$count ~ deliveries$time, type="p",
         main="Cumulative number of deliveries",
         xlab="Time of day",
         ylab="Cumulative number of deliveries",
         scales=list(x=list(tick.number=6, cex=0.6)),
         xlim=prettyEdges ),
  "ndel.png", h=4, w=6 )
#with graphics ndel.png timeout 120
```



What do the wait times look like?

```
<R60> mp(
  xyplot( getDur/60/60 ~ getPTime, data=d,
  main="File delivery waits",
  xlab="Time of day of request", xlim=prettyEdges,
  ylab="Wait time (hours)",
  scales=list(x=list(tick.number=6, cex=0.6))
),
"getWaits.png", h=4, w=6
)
#with graphics getWaits.png timeout 60
```

## File delivery waits



```
<R61> mp(
  xyplot( getDur/60/60 ~ delPTime, data=d,
    main="File delivery waits",
    xlab="Time of day of delivery", xlim=prettyEdges,
    ylab="Wait time (hours)",
    scales=list(x=list(tick.number=6, cex=0.6))
  ),
  "delWaits.png", h=4, w=6
)
#with graphics delWaits.png timeout 60
```

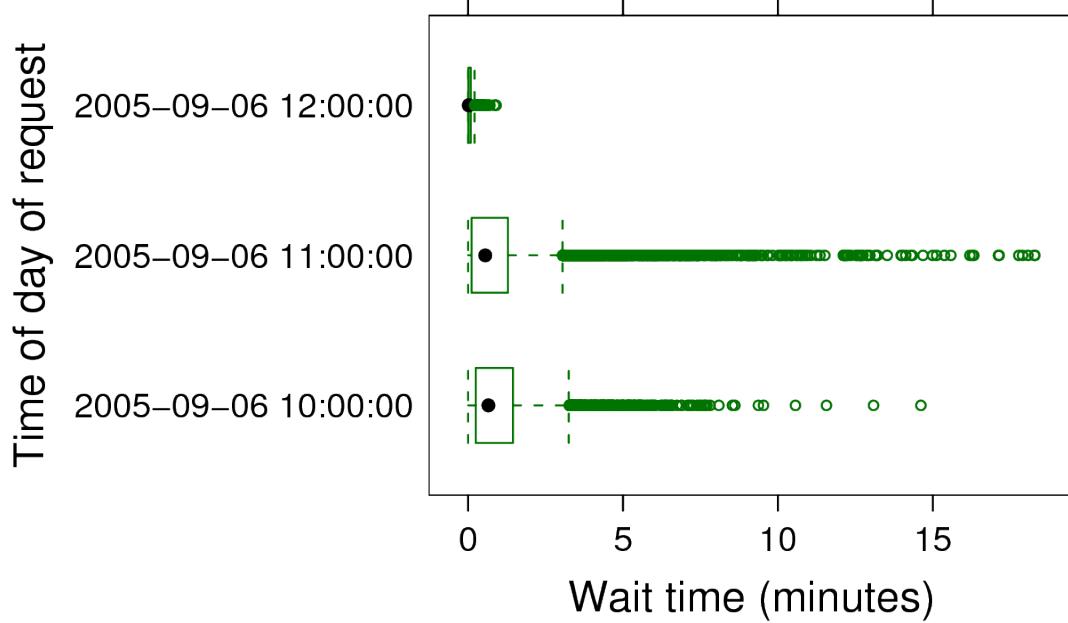
## File delivery waits



These plots make the outliers really stand out and hard to see the mean wait times. Let's try to plot those...

```
<R63> mp(
  bwplot( getsPerHourCuts ~ getDur/60, data=d,
    main="File delivery waits",
    xlab="Wait time (minutes)", ylab="Time of day of request"
  ),
  "waitsPerHourBw.png", h=4, w=6
)
#with graphics waitsPerHourBw.png timeout 60
```

## File delivery waits



Again, the outliers dominate the plot. Let's just plot medians and how many are over an hour...

```
<R64> cleanNA = function(x) x[ ! is.na(x) ]  
  
<R65> waitMedians = tapply(d$getDur/60, getsPerHourCuts, median,  
    na.rm=T)  
  
<R66> waitMedians = waitMedians[ ! is.na(waitMedians) ]  
  
<R67> waitMeans = tapply(d$getDur/60, getsPerHourCuts, mean) ;  
    waitMeans = cleanNA(waitMeans)  
  
<R68> nOverHour = tapply(d$getDur/60/60 >= 1, getsPerHourCuts, sum) ;  
    nOverHour = cleanNA(nOverHour)  
  
<R69> getsPerHour = getsPerHour[ getsPerHour > 0 ]  
  
<R70> percOverHour = nOverHour / getsPerHour * 100  
  
<R71> nOverHalfHour = tapply(d$getDur/60/60 >= 0.5, getsPerHourCuts,  
    sum) ; nOverHalfHour = cleanNA( nOverHalfHour)
```

```
<R72> percOverHalfHour = nOverHalfHour / getsPerHour * 100

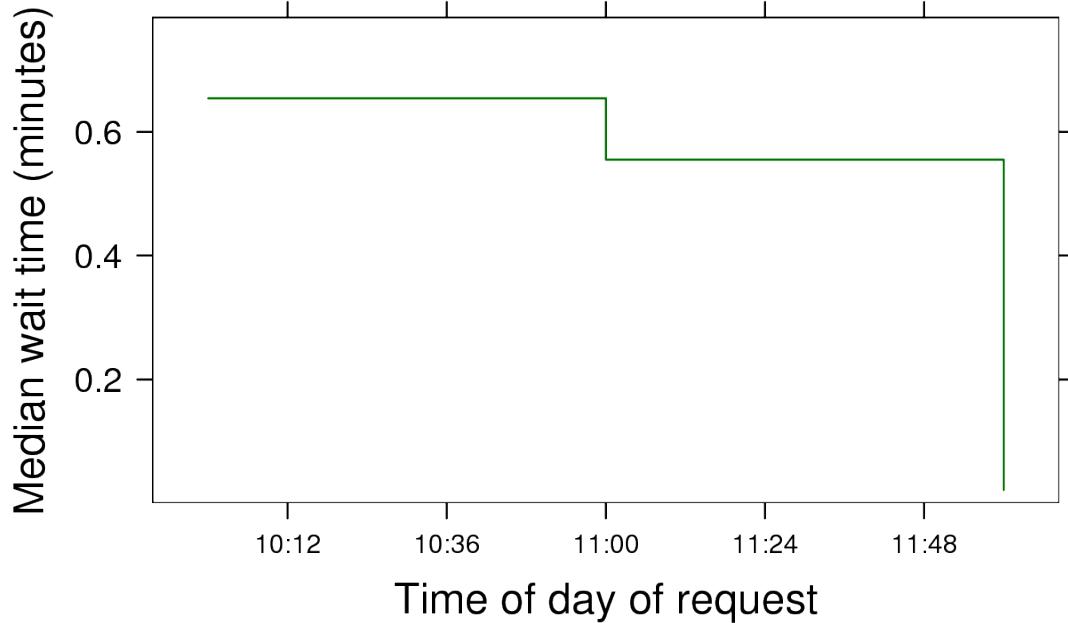
<R73> nUnderMinute = tapply(d$getDur/60 < 1, getsPerHourCuts, sum) ;
      nUnderMinute = cleanNA( nUnderMinute)

<R74> percUnderMinute = nUnderMinute / getsPerHour * 100
```

Let's plot these things

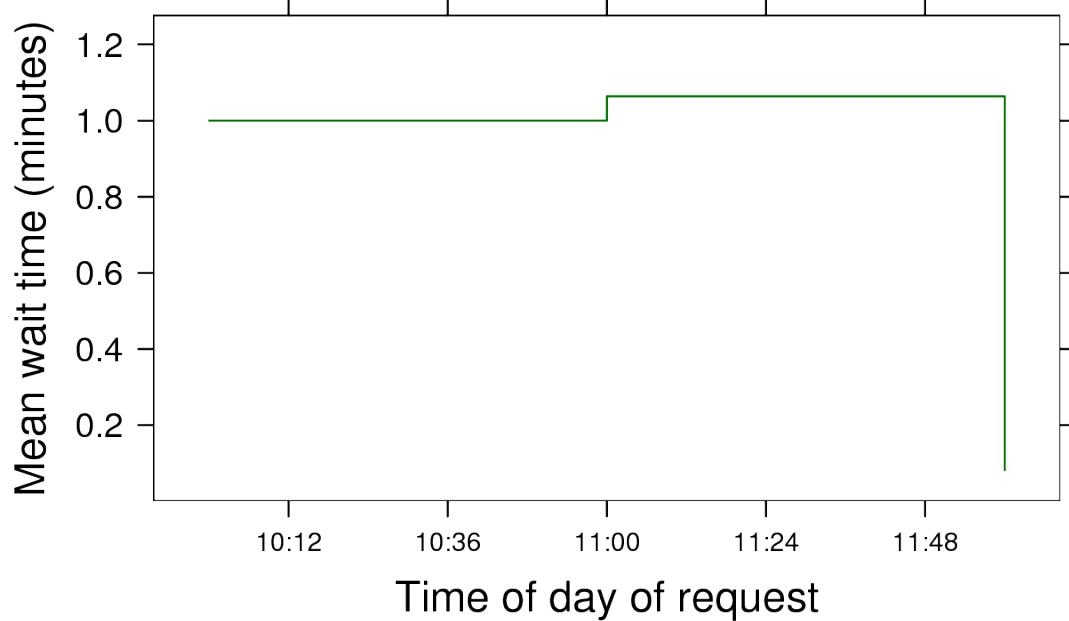
```
<R75> mp(
  xyplot( waitMedians ~ as.POSIXct(names(waitMedians)), type="s",
         main="Median wait time", xlab="Time of day of request",
         ylab="Median wait time (minutes)",
         ylim=c(0, max(waitMedians)*1.2),
         scales=list(x=list(tick.number=6, cex=0.6)))
),
"medians.png", h=4, w=6)
#with graphics medians.png timeout 60
```

## Median wait time



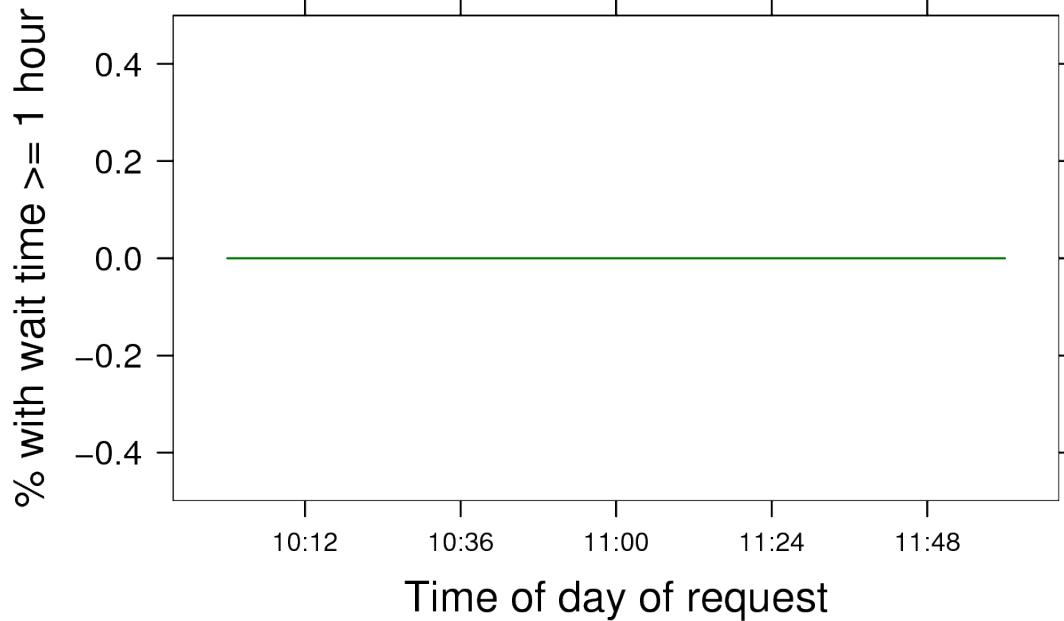
```
<R76> mp(
  xyplot( waitMeans ~ as.POSIXct(names(waitMeans)), type="s",
         main="Mean wait time", xlab="Time of day of request",
         ylab="Mean wait time (minutes)",
         ylim=c(0, max(waitMeans)*1.2),
         scales=list(x=list(tick.number=6, cex=0.6))
  ),
  "means.png", h=4, w=6)
#with graphics means.png timeout 60
```

## Mean wait time



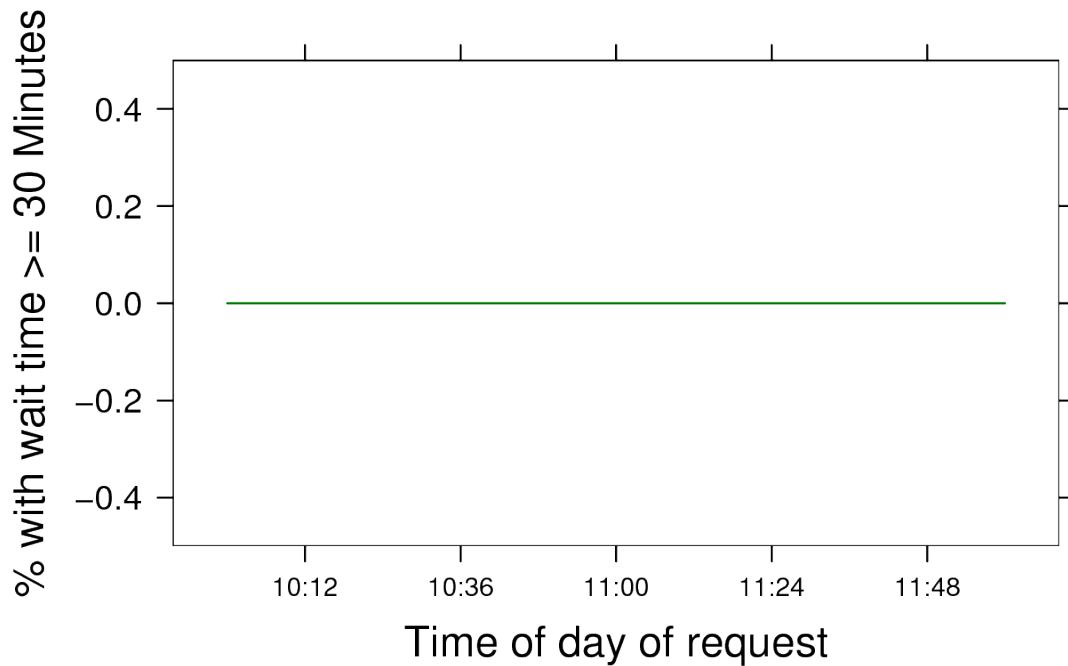
```
<R77> mp(
  xyplot( percOverHour ~ as.POSIXct(names(percOverHour)), type="s",
         main="Percent long wait times", xlab="Time of day of
request",
         ylab="% with wait time >= 1 hour",
         scales=list(x=list(tick.number=6, cex=0.6))
  ),
  "overHour.png", h=4, w=6)
#with graphics overHour.png timeout 60
```

## Percent long wait times



```
<R78> mp(
  xyplot( percOverHalfHour ~ as.POSIXct(names(percOverHalfHour)),
  type="s",
    main="Percent long wait times", xlab="Time of day of
request",
    ylab="% with wait time >= 30 Minutes",
    scales=list(x=list(tick.number=6, cex=0.6))
),
"overHalfHour.png", h=4, w=6)
#with graphics overHalfHour.png timeout 60
```

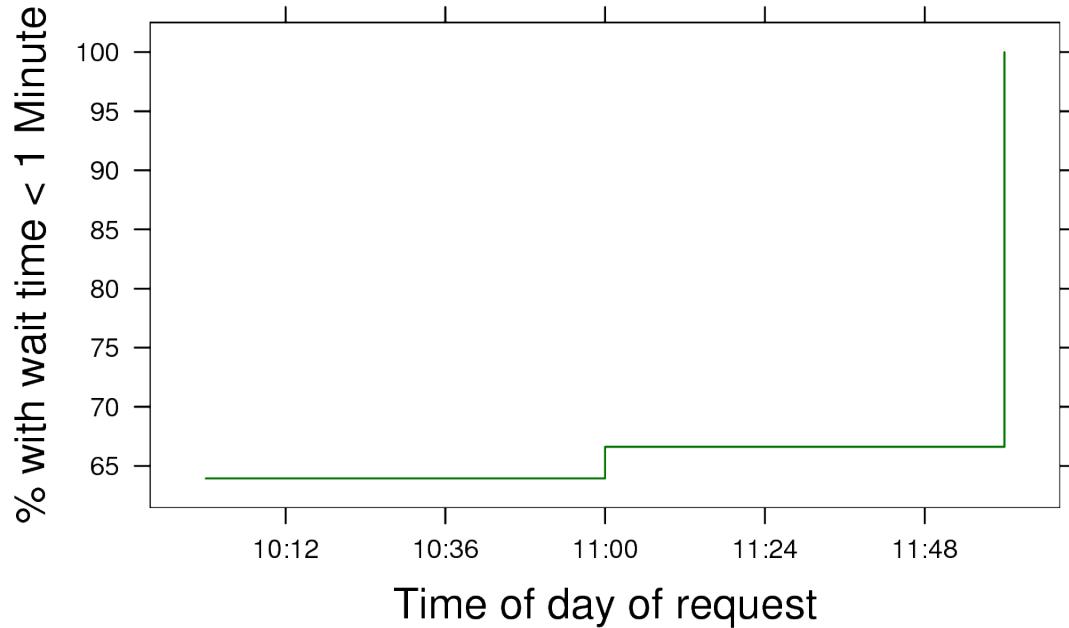
## Percent long wait times



Look for short wait times.

```
<R79> mp(
  xyplot( percUnderMinute ~ as.POSIXct(names(percUnderMinute)),
  type="s",
    main="Percent short wait times", xlab="Time of day of
request",
    ylab="% with wait time < 1 Minute",
    scales=list(tick.number=6, cex=0.6)
  ),
  "underMinute.png", h=4, w=6)
#with graphics underMinute.png timeout 60
```

## Percent short wait times



### A.2.5 Look at number of open requests

Let's look at the number of instantaneous requests that are open at any given time.

Get next file requests...

```
<R80> gnf = data.frame( time=d$getPTime, adj=1 )
```

File deliveries (request fulfilled)

```
<R81> gnc = data.frame( time=d$delPTime, adj=-1 )
```

Join them,

```
<R82> gn = rbind(gnf, gnc)
```

Sort by time,

```
<R83> gn = gn[ order(gn$time), ]
```

Do a running count of # of unfulfilled get next file requests

```
<R84> gn$count = cumsum(gn$adj)
```

```
<R85> gn[1:10,]
```

		time	adj	count
1881	2005-09-06	10:27:14	1	1
6041	2005-09-06	10:27:14	1	2
3161	2005-09-06	10:27:16	1	3
281	2005-09-06	10:27:20	1	4
2521	2005-09-06	10:27:22	1	5
14721	2005-09-06	10:27:23	1	6
7001	2005-09-06	10:27:28	1	7
4761	2005-09-06	10:27:30	1	8
1241	2005-09-06	10:27:31	1	9
15641	2005-09-06	10:27:42	1	10

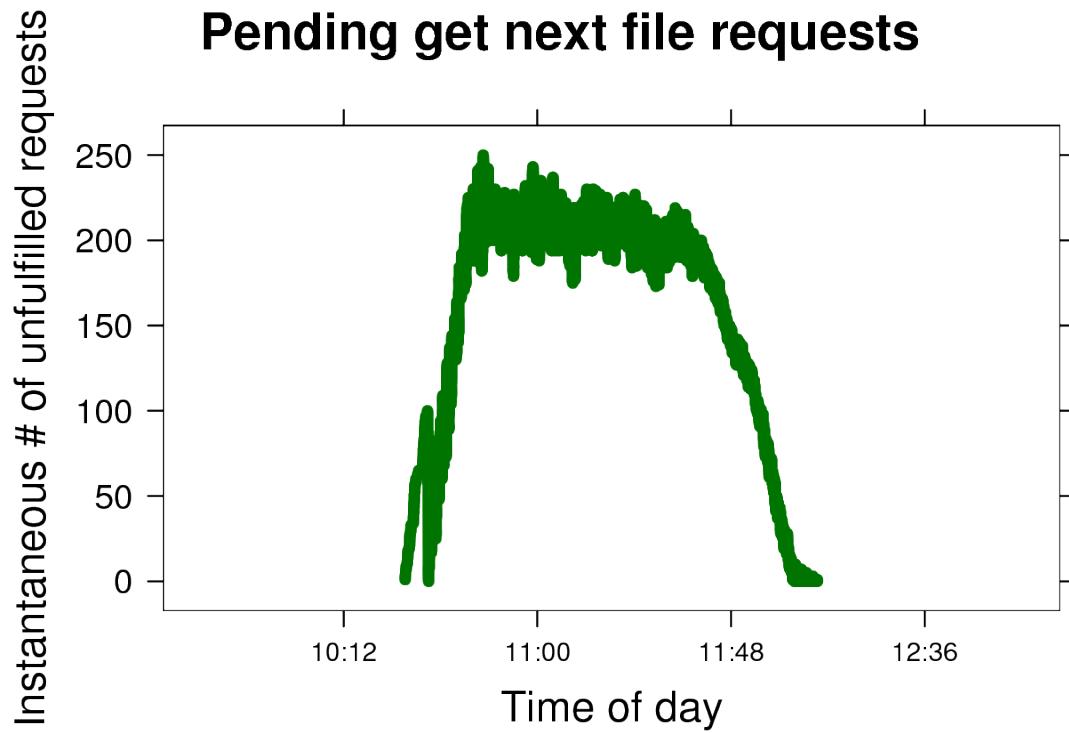
<R86> summary(gn\$count)

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.0	140.0	198.0	169.6	211.0	250.0

Plot it...

<R87> mpC

```
xyplot( gn$count ~ gn$time, type="p",
        main="Pending get next file requests",
        xlab="Time of day",
        ylab="Instantaneous # of unfulfilled requests",
        scales=list(x=list(tick.number=6, cex=0.6)),
        xlim=prettyEdges, pex=0.1 ),
"unfulfilled.png", h=4, w=6 )
#with graphics unfulfilled.png timeout 240
```



I think this just shows the affect of the request wait time being longer than the request rate.